PR #36949 完整报告

vllm-project/vllm

[ROCm][CI] Optimize ROCm Docker build: registry cache, DeepEP, and ci-bake script

合并时间：2026-06-03 14:43

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/36949

执行摘要

- 一句话：分层缓存加速 ROCm Docker 构建
- 推荐动作：建议 CI 和基础设施团队精读，重点关注构建缓存分层策略、ccache vs sccache 选择、artifact 模式设计。对于仅关注算法和模型的开发者可略过。

功能与动机

当前每个 PR 都从头构建 RIXL、DeepEP、rocshmem、torchcodec 和 RDMA 库，平均耗时 26 分钟。本 PR 引入预构建的 ci_base 镜像来吸收这些稳定层，使每 PR 构建只需构建薄的 vLLM wheel 和 workspace 层，极大减少等待时间。同时解决了 triton_kernels 编译问题，并通过 content-addressed 缓存避免不必要的层重建。

实现拆解

1. 新增统一构建脚本(ci-bake-rocm.sh, +1749 行)：封装 Docker buildx bake 调用，处理缓存分支标签、内容哈希比较、构建器管理。核心函数 compose_cache_branch_tag 生成带架构哈希的稳定标签，select_cache_branch_name 按优先级选择分支名。
2. 新增 Bake 配置文件(docker-bake-rocm.hcl 和 ci-rocm.hcl)：定义 test-rocm、ci_base 等构建目标及 CI 专用变量（如 CI_BASE_IMAGE、缓存键变量）。ci-rocm.hcl 作为 vLLM 仓库内的覆盖层，使构建逻辑随 repo 演进。
3. 重构 Dockerfile.rocm：引入 csrc-build-rocm 阶段独立编译 HIP 内核（避免 Python 变更触发重编），添加 ccache/mold 加速链接，增加 CI_BASE_IMAGE 参数使 test 阶段从预构建镜像继承。新增 DeepEP wheel 安装和 rocshmem 运行时拷贝。
4. 更新 CI 流水线(amd.yaml)：将原先的单步构建拆分为 ensure-ci-base 和 build test image 两步。ensure-ci-base 通过内容哈希检查决定是否重建 ci_base；build test image 支持 ROCM_CI_ARTIFACT_ONLY 模式从 artifact 构建测试镜像。
5. 新增 CI 配置(ci_config_rocm.yaml)：定义触发全 CI 匹配的模式，包括 Dockerfile.rocm、ci-bake-rocm.sh 等文件。
6. 配套脚本改进(run-amd-test.sh)：增加 prepare_artifact_image 函数，从 buildkite artifact 下载 wheel 包并构建本地测试镜像；install_torchcodec_rocm.sh 添加 sccache 支持和错误处理。

关键文件：

- .buildkite/scripts/ci-bake-rocm.sh（模块 构建脚本；类别 other；类型 dependency-wiring；符号 add_arg, docker_hub_repo, compose_cache_branch_tag, select_cache_branch_name）：核心新脚本，封装了 Docker buildx bake 调用、缓存标签生成、内容哈希检查、构建器管理，是本次 PR 的关键复杂性所在。
- docker/ci-rocm.hcl（模块 Docker 配置；类别 infra；类型 infrastructure）：CI 专用 Bake 配置覆盖层，定义 CI_BASE_IMAGE、缓存键等变量，使构建参数随 repo 演进，是分层构建的关键配置。
- .buildkite/hardware_tests/amd.yaml（模块 CI 配置；类别 test；类型 test-coverage）：定义 ROCm CI 流水线步骤，新增 ensure-ci-base 和 artifact-only 路径，直接影响 CI 行为。
- docker/docker-bake-rocm.hcl（模块 Docker 配置；类别 infra；类型 infrastructure）：ROCm 专用 Bake 主配置文件，定义分组、公共变量和目标，与 ci-rocm.hcl 配合形成完整构建规范。
- docker/Dockerfile.rocm（模块 Docker 构建；类别 infra；类型 infrastructure）：实际 Dockerfile 变更，引入新构建阶段、ccache/mold、DeepEP 安装、CI_BASE_IMAGE 参数等，是构建逻辑的核心载体。
- .buildkite/scripts/hardware_ci/run-amd-test.sh（模块 测试脚本；类别 infra；类型 infrastructure）：测试执行脚本，新增 artifact 映像准备功能，支持从构建产物构建测试环境。

关键符号：add_arg, docker_hub_repo, compose_cache_branch_tag, select_cache_branch_name, cache_scope_suffix, prepare_artifact_image

关键源码片段

docker/ci-rocm.hcl

CI 专用 Bake 配置覆盖层，定义 CI_BASE_IMAGE、缓存键等变量，使构建参数随 repo 演进，是分层构建的关键配置。

```
# ci-rocm.hcl - CI-specific configuration for vLLM ROCm Docker builds
#
# This file lives in the vLLM repo at docker/ci-rocm.hcl so ROCm Docker
# build mechanics can evolve with Dockerfile.rocm and docker-bake-rocm.hcl.
# Used with: docker buildx bake -f docker/docker-bake-rocm.hcl -f docker/ci-rocm.hcl test-rocm-ci
#

# CI metadata

variable "BUILDKITE_COMMIT" {
  default = ""
}

variable "BUILDKITE_BUILD_NUMBER" {
  default = ""
}
```

```
variable "BUILDKITE_BUILD_ID" {
  default = ""
}

variable "PARENT_COMMIT" {
  default = ""
}

variable "VLLM_MERGE_BASE_COMMIT" {
  default = ""
}

variable "COMMIT" {
  default = BUILDKITE_COMMIT
}

# Image tags (set by CI)

variable "IMAGE_TAG" {
  default = ""
}

variable "IMAGE_TAG_LATEST" {
  default = ""
}

# ROCm-specific GPU architecture targets

variable "PYTORCH_ROCM_ARCH" {
  default = "gfx90a;gfx942;gfx950"
}

# Pre-built CI base image (Tier 1). Per-PR builds pull this instead of
# rebuilding RIXL/DeepEP/torchcodec from scratch.
variable "CI_BASE_IMAGE" {
  default = "rocm/vllm-dev:ci_base"
}

variable "CI_MAX_JOBS" {
  default = ""
}

# Upstream dependency commit pins
variable "RIXL_BRANCH" { default = "" }
variable "UCX_BRANCH" { default = "" }
variable "ROCSHMEM_BRANCH" { default = "" }
variable "DEEPEP_BRANCH" { default = "" }
variable "RIXL_CACHE_KEY" { default = "" }
variable "ROCSHMEM_CACHE_KEY" { default = "" }
```

variable "DEEPEP_CACHE_KEY" { default = "" }

评论区精华

- 内容哈希标签问题（mawong-amd）：质疑 ci_base 使用固定标签 rocm/vllm-dev:ci_base 可能导致 main 和 PR 分支间的标签竞争和缓存失效。作者未直接回复，但 CI 实践中已通过内容哈希比较决定是否重建，减少了重建频率。
- rocshmem 依赖遗漏（gemini-code-assist 两次 critical）：指出 test 和 final 阶段安装 DeepEP wheel 但未拷贝 rocshmem 运行时库。作者回复“Done :)”并在后续提交中修复。
- REMOTE_VLLM 下主机文件拷贝（gshtras）：质疑 csrc-build-rocm 阶段直接拷贝本地文件违背 REMOTE_VLLM 设计。作者回复进行了重构，恢复传统方式并引入按架构构建。
- Docker builder 命名不一致（gemini-code-assist）：ci-bake-rocm.sh 中硬编码 baked-vllm-builder 与可定制 BUILDER_NAME 冲突。作者回复“Done :)”后统一使用变量。
- 安全校验缺失（depthfirst-app）：CI_HCL_SOURCE 从 URL 下载 HCL 文件无完整性校验，建议添加 SHA256 检查。当前尚未解决，属低风险。
- ci_base 内容哈希标签竞争 (design): 作者未直接回复，但后续实现中已通过内容哈希比较决定是否重建，标签仍固定但重建频率降低。风险未完全消除。
- rocshmem 运行时依赖丢失 (correctness): 作者回复“Done :)”并在后续提交中恢复 COPY 指令。
- REMOTE_VLLM 下主机文件拷贝 (design): 作者回复重构了该阶段，恢复传统方式并引入按架构构建，避免远程模式下拷贝本地文件。
- 安全下载无完整性校验 (security): 作者未回复，该问题未解决，属于低风险但需后续改进。

风险与影响

- 风险：
  1. ci_base 标签竞争：若未使用内容哈希标签，main 与 PR 分支可能同时覆盖 rocm/vllm-dev:ci_base，导致缓存不一致。虽已通过内容哈希比较减少重建，但标签本身仍为固定字符串，仍存在竞争窗口。
  2. rocshmem 依赖遗漏：早期版本在 test/final 阶段未拷贝 rocshmem 运行时，可能导致 DeepEP 运行时错误。已修复，但需警惕类似依赖遗漏。
  3. 缓存后端单点：当前仅使用 Docker Hub 作为 BuildKit 层缓存，无 S3 共享缓存，若 Hub 不稳定则构建可能退化。
  4. 安全下载无校验：ci-bake-rocm.sh 支持从 URL 加载 HCL 文件，但无 SHA256 验证，可能被 MITM 攻击。
  5. 构建器命名混淆：脚本中硬编码 builder 名称与自定义变量不一致，已修复。
  6. 新代码复杂度：1749 行的新脚本增加了维护负担，且与 ci-infra 仓库脚本需保持同步。 - 影响：对用户无直接影响（纯 CI 变更）。对团队：显著降低 ROCm CI 平均构建时间（从 ~26 分钟降至数分钟），提高迭代效率；artifact 模式使测试节点无需重复编译，资源消耗减少。新增维护成本：ci_base 镜像需定期重建（建议每周），内容哈希文件列表变更时需同步更新 ci-rocm.hcl 中的 CI_BASE_CONTENT_FILES。 - 风险标记：ci_base 标签竞争，rocshmem 依赖遗漏，缓存后端单点，安全下载无校验，构建器命名

混淆，新代码复杂度

关联脉络

- 暂无明显关联 PR